2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

What is new in IP-XACT Std. IEEE 1685-2022?

Richard Weber (Arteris IP)
Edwin Dankert (ARM)

accellera SYSTEMS INITIATIVE

Outline

- What is IP-XACT?
- Changes in IP-XACT IEEE Std. 1685-2022 – Part 1
- Questions
- Changes in IP-XACT IEEE Std. 1685-2022 – Part 2
- Questions
- Questions for the audience

IP-XACT, a standard structure for packaging, …

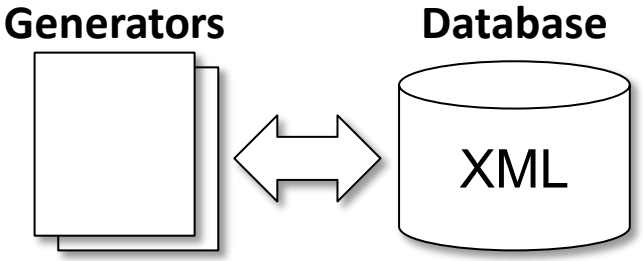
- XML Schema Definition (XSD)
  - component, design, …
- Semantic Consistency Rules (SCR)
  - overlapping registers, multiple drivers, …
- Tight Generator Interface (TGI)
  - API provided by IP-XACT Design Environment

"Electronic IP Data Sheet"

- Bus interfaces
- Ports
- Registers
- Parameters
- Views
- Files

"Electronic IC Data Sheet"

- IP instances
  - Parameter values
- Interconnect



IP-XACT development

• IP-XACT 1.0	December 2004
• IP-XACT 1.1	June 2005
• IP-XACT 1.2	April 2006
• IP-XACT 1.4	March 2008
• IEEE Std. 1685-2009	December 2009
• IEEE Std. 1685-2014	June 2014
• IEEE Std. 1685-2022	September 2022

Changes in IEEE Std. 1685-2022 (Part-1)

- Removed conditionality
- Added XML document to describe memory element definitions
- Added mode-dependent memory and register access
- Added mapping from ports to register fields
- Added register field aliasing and broadcasting
- Added connectivity support for structs, unions, and SV interfaces
- Added power domains and power domain bindings
- Added REST as standard transport layer for TGI

Removed conditionality

- To reduce complexity that was introduced in IEEE Std. 1685-2014
- Backward compatibility supported by Accellera Vendor Extensions

```
<ipxact:port>
  <ipxact:name>my_port</ipxact:name>
  …
  <ipxact:isPresent>
    …some Boolean expression…
  </ipxact:isPresent>
<ipxact:port>
```

```
<ipxact:port>
  <ipxact:name>my_port</ipxact:name>
  …
  <ipxact:vendorExtensions>
    <accellera:port>
      <accellera-cond:isPresent>
       …some Boolean expression…
      </accellera-cond:isPresent>
    </accellera:port>
  <ipxact:vendorExtensions>
<ipxact:port>
```

Added typeDefinitions

- New top-level element
- Describes memory-related element definitions
  - Field definition, register definition, address block definition, …
- Instantiated in components and other type definitions

Example typeDefinitions

typeDefinitions

```
VLNV=abcd
registerDefinition
        name=R1
        size=32
        …
registerDefinition
        name=R2
        …
parameters P1, P2, P3
```

component

```
VLNV=efgh
externalTypeDefinitions
        name=myTypes
        VLNV ref=abcd
                configurableElements P1=1, P2=2, P3=3
memoryMap
        addressBlock
                register
                        name=myReg
                        addressOffset=0
                        registerDefinitionRef=R1, typeDefinitions=myTypes
```

Component modes

- New component sub-element **mode**
- Describes user-defined modes and conditions when modes are active
- Modes are referenced in memory elements, ports, and interfaces, e.g
  - memoryRemap (replacing remapState)
  - alternateRegister (replacing alternateGroup)
  - addressBlock, registerFile, and register accessPolicy (new)
  - field fieldAccessPolicy (new)
  - field accessRestriction (new)
  - port fieldMap (new)

Example mode-dependent register access

component modes

```
mode
        name=TEST
        portSlice=myPortSlice
                portRef=test_mode
        condition=$ipxact_port_value(myPortSlice)==1
mode
        name=FUNC
        portSlice=myPortSlice
                portRef=test_mode
        condition=$ipxact_port_value(myPortSlice)==0
```

register access policy

```
register
        name=myReg
        addressOffset=0
        size=32
        accessPolicy
                modeRef=TEST
                access=read-write
        accessPolicy
                modeRef=FUNC
                access=read-only
```

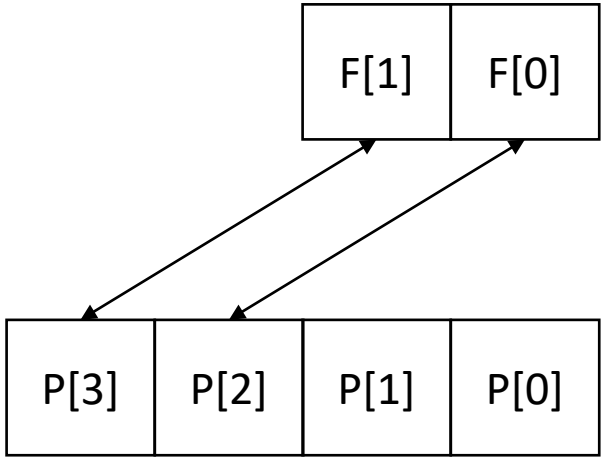
A condition expression can contain terms for port slice values, register field slice values, and mode condition values

Field map

- New port sub-element **fieldMap** (mode-dependent)
- describes if a field slice is accessible through a port slice in the referenced modes

Example field map

```
register
        name=R
        field
                name=F
                bitOffset=0
                bitWidth=2
port
        name=P
        vector=[3:0]
        fieldMap
                fieldSlice
                        registerRef=R
                        fieldRef=F
                range=[3:2]
                modeRef=FUNC
```



Register field aliasing and broadcasting

- New register field sub-element **aliasOf** (mode-independent)
  - describes if a field is an alias of another field in the same component
- New register field sub-element **broadcastTo** (mode-dependent)
  - describes if a write transaction on a field triggers the same transaction on other fields in the same component in the referenced modes

Example field aliasing and broadcasting

alias

```
register
        name=R1
        addressOffset=0
        field
                name=F
                access=read-only
register
        name=R2
        addressOffset=4
        field
                name=F
                access=read-write
                aliasOf
                        registerRef=R1
                        fieldRef=F
```

broadcast

```
register
        name=R1
        addressOffset=0
        field
                name=F
                access=write-only
register
        name=R2
        addressOffset=4
        field
                name=F
                access=read-write
                broadcastTo
                        registerRef=R1
                        fieldRef=F
```

Structured port

- New component port sub-element **structured**
- Describes an HDL port that is a structure, union, or SV interface that is composed out of binary values
- Connections can be made to the whole HDL port or to slices of it

Example structured port – structure

structure

```
name=myPort
structured, packed=true
        struct, direction=out
        subPort
                name=mySubPort1
                wire
                        direction=out
                        vector=[11:0]
        subPort
                name=mySubPort2
                wire
                        direction=out
                        vector=[3:0]
```

```
typedef struct packed {
        logic [11:0] mySubPort1;
        logic  [3:0] mySubPort2;
} myStruct;

module m(output myStruct myPort);
endmodule
```

Example structured port – SV interface

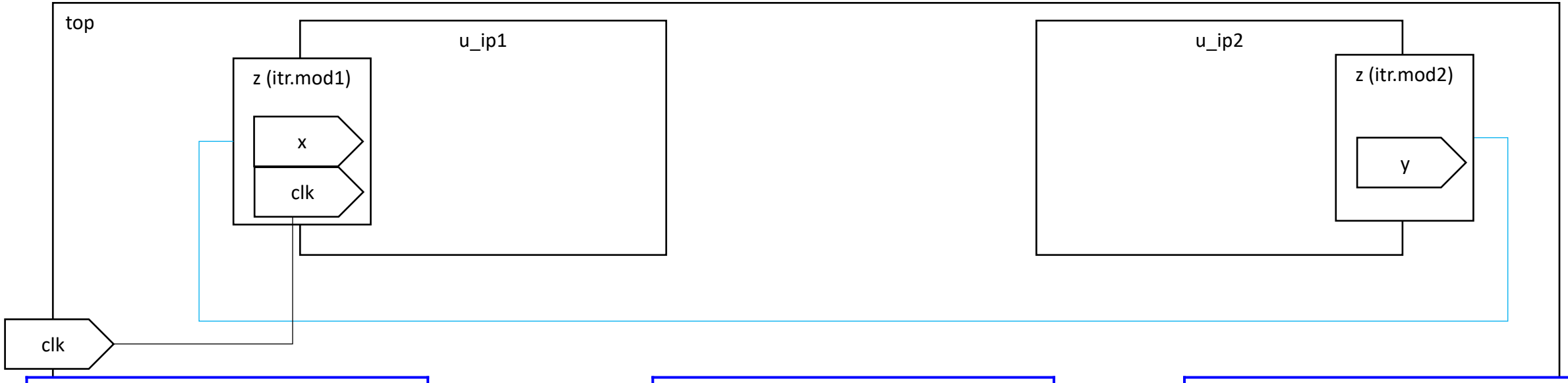
SV interface

```
name=z
structured
        interface
        subPort, isIO=true
                name=clk
                wire
                        direction=in
        subPort
                name=x
                wire
                        direction=inout
        structPortTypeDef
                typeName=itr
                role=mod1
```

```
interface itr( input clk );
  logic x,y;
  modport mod1( input x, clk );
  modport mod2( output y );
endinterface


module ip( itr.mod1 z );
endmodule
```

Example structured port – SVI connection




```
interface itr( input clk );
  logic x,y;
  modport mod1( input x, clk );
  modport mod2( output y );
endinterface

module ip1( itr.mod1 z );
endmodule
```

```
module top(input clk);
  itr my_if(.clk(clk));
  ip1 u_ip1(.z(my_if));
  ip2 u_ip2(.z(my_if));
endmodule
```

```
interface itr( input clk );
  logic x,y;
  modport mod1( input x, clk );
  modport mod2( output y );
endinterface

module ip2( itr.mod2 z );
endmodule
```

Power domains

- New component sub-element **powerDomain**
  - Describes power domains of the component
- New component port sub-element **powerDomainRef**
  - Describes in which power domain the port is located
- New design component instance sub-element **powerDomainLinks**
  - Describes binding of component instance power domains

Example power domains

component

```
VLNV=abcd
powerDomain
        name=D1
powerDomain
        name=D2
```

component

```
VLNV=efgh
powerDomain
        name=T1
powerDomain
        name=T2
designInstantiation
        VLNV ref=wxyz
```

design

```
VLNV=wxyz
componentInstance
        instanceName=myInstance
        VLNV ref=abcd
        powerDomainLink
                externalPowerDomainReference=T1
                internalPowerDomainReference=D1
        powerDomainLink
                externalPowerDomainReference=T2
                internalPowerDomainReference=D2
```

Questions Part-1

Changes in IEEE Std. 1685-2022 (Part-2)

- Updated terminology
- Updated compatibility/mapping/connectivity rules
- Added bus-definition packets
- Added CPU memory-map (+ updated addressing rules)
- Added more descriptions / documentation and escape sequences
- Added ipxact functions
- Qualifiers on ports

Updated terminology

- Changed interface roles to Initiator and Target
- Introducing clearbox

Updated compatibility/mapping/connectivity rules

- Added match to indicate:
  - If this element is enabled (true) it is an error if the port is not mapped on both sides of the connection.
- Added allBits attribute on wirePort width:
  - false: mapping is optional, any number of bits can be mapped.
    true: mapping is required the full width is supposed to be mapped.

Updated compatibility/mapping/connectivity rules - Table

Port	Width	Match	Description	Use case
required	True	-	Port is required to be mapped, the width needs to match the width specified in the abs-def.	AHB5.HSIZE
required	False	-	Port is required to be mapped, the width does not need to match (and does not need to match the width specified in the abs-def)	APB.PADDR
optional	True	False	Port is optional to be mapped, the width needs to match the width specified in the abs-def.	AXI5.BRESP
optional	True	True	Port is optional to be mapped but needs to match on both sides of the connection, the width needs to match the width specified in the abs-def.	APB5.PREADYCHK
optional	False	False	Port is optional to be mapped, the width does not need to match (and does not need to match the width specified in the abs-def)	AXI5.BUSER
optional	False	True	Port is optional to be mapped but needs to match on both sides of the connection, the width does not need to match (and does not need to match the width specified in the abs-def)	???

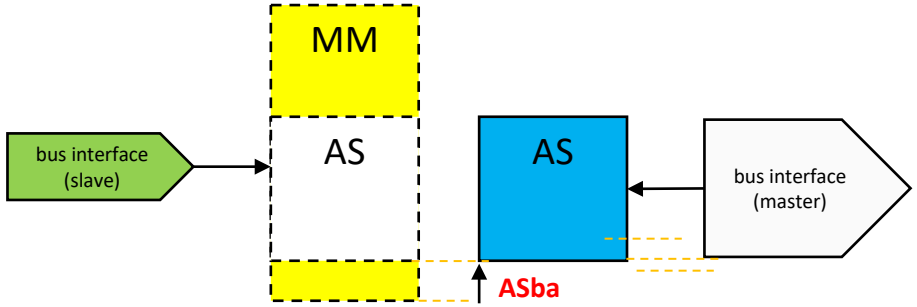
Added abstractionDefinition port packets

- Provide additional packet information for the transactional or wire port types.
- Allows for multiple packets and one or more packetFields per packet.
- Allowing to provide a more complete description of the underlying protocol.

Updated CPU memory-map

- Previously unable to model the CPU memory-map fully.
- Removed address-space ref from CPU and replaced with a set of regions and a memory-map reference.
- Allowing to capture the predefined address regions and the target interface mappings

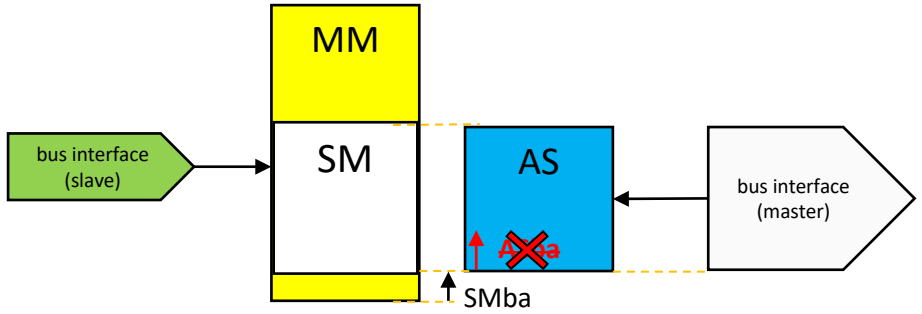
Addressing Rules Simplification



No subspaceMap

- opaque and transparent behave the same
- ASba + AS@

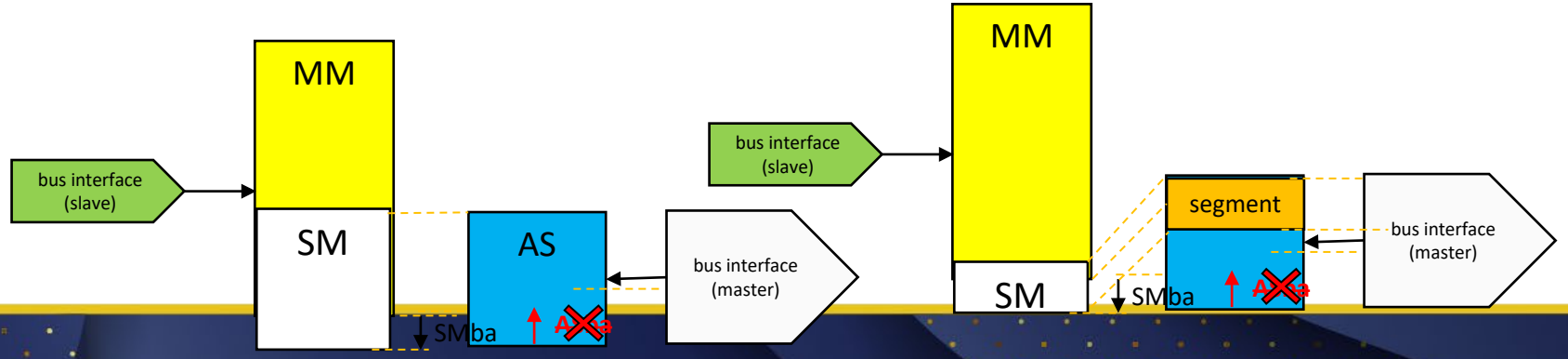
Note: ASba=addressSpaceRef.baseAddress



SubspaceMap (SM)

- SMba + AS@
- ASba is ignored

Note: SMba=SubspaceMap.baseAddress



Note: requires SubspaceMap.baseAddress to be of a signed integer type

UG Updates (12. Addressing)

$$bridge_master_bit_address = bridge_master_bus_address^{(37)} \times bridge_master.bitsInLau + \cancel{bridge_master.addressSpaceRef.baseAddress \times addressSpace.addressUnitBits} \quad (38)$$

1) For a transparent bridge:

$$bridge_slave_address = bridge_address_space_address^{(39)} \times addressSpace.addressUnitBits / bridge_slave.bitsInLau + \quad (41)$$

$$\boxed{bridge_master.addressSpaceRef.baseAddress \times addressSpace.addressUnitBits}$$

2) For an opaque bridge:

$$bridge_slave_address = (((bridge_address_space_address^{(39)} - segment.addressOffset) \times addressSpace.addressUnitBits) / bridge_slave.bitsInLau) + slave_bus_address^{(26)} \quad (42)$$

Note: the addressSpaceRef.baseAddress value is only applied to the

Added more descriptions / documentation

- Added displayName to top level elements (component / busDefinition / etc …)
- Added shortDescription
- Specified escape sequences for use in displayName, description and shortDescription
  - To be resolved by DE or generator

Escape sequence	Description
$ipxact_index_value (identifier)	IP-XACT DE or generator resolvable string value representing the array index from the array dimension that has the indexVar attribute value matching the argument.
$ipxact_parameter_value (identifier)	IP-XACT DE or generator resolvable string value representing the parameter that has the parameterId attribute value matching the argument

Added IP-XACT specific functions

Function	Allowed	Description
$ipxact_absdefport_value (identifier)	packetfield/width	Returns an IP-XACT unresolved bit expression value representing an abstraction definition port value.
$ipxact_field_value (identifier)	condition	Returns an IP-XACT unresolved bit expression value representing a field slice value.
$ipxact_index_value (identifier)	pathSegment	Returns an IP-XACT unresolved unsigned longint expression value representing the array index from the array dimension that has the indexVar attribute value matching the argument
$ipxact_mode_condition (identifier)	condition	Returns an IP-XACT unresolved bit expression value representing a mode condition value.
$ipxact_packetfield_value (identifier)	packetfield/width	Returns an IP-XACT unresolved bit expression value representing a packet field value.
$ipxact_port_value (identifier)	condition	Returns an IP-XACT unresolved bit expression value representing a component port slice value

- *Note: also added $sformatf supporting elements of type stringExpression (pathSegments)*

Qualifiers on wire ports

(abstractionDefinition + Component)

Name	Description
isValid	Specifies the data on the interface is currently valid
isInterrupt	Specifies the port contains interrupt information
isClockEn (@level)	Specifies the port contains clock enable information. isClockEn has a level attributes similar to isReset. The level attribute describes the assertion level of the clock enable information.
isPowerEn (@level, @powerDomainRef)	Specifies the port contains power enable information. The level attribute describes the assertion level of the power enable information. The powerDomainRef attribute references a power domain of the encapsulating component.
isOpcode	Specifies the port contains operational code or opcode information.
isProtection	Specifies that the signal is used by a protection or security mechanism
isFlowControl (@flowType, @user)	Specifies that the port contains flow control information. The flowType attribute describes additional information on the type of flow control. The user attribute can be used to provide user-defined flow control types.
isRequest	Specifies the port contains request information
isResponse	Specifies the port contains response information
isUser (@user)	Specifies the port contains user-defined information. The user attributed can be used to describe user-defined information.

Questions Part-2

Questions for the audience

- Do you plan to use / adopt 1685-2022 in the near future?
- What new functionality / changes are you planning to use in your internal flows
- Which of these new features do you hope IP providers implement
- Which of this new functionality do you hope EDA tool providers provide implementations for